Contraintes mémoire et solution architecturale pour applications TDSI

G. CORRE, N. JULIEN, E. SENN, E. MARTIN

LESTER, Université de Bretagne Sud, centre de recherche, BP 92116 - 56321 LORIENT Cedex

gwenole.corre@univ-ubs.fr
nathalie.julien@univ-ubs.fr
eric.senn@univ-ubs.fr
eric.martin@univ-ubs.fr

Résumé – Les systèmes supportant des applications de traitement du signal et de l'image manipulent de plus en plus de données. Cela entraîne une utilisation intensive de la mémoire qui devient le point critique du système ; la mémoire limite les performances et représente une proportion importante de la consommation globale. Dans le cadre du projet RNRT ALIPTA nous développons l'outil de synthèse d'architecture GAUT en nous intéressant à la synthèse de la partie mémoire. Nous évaluerons l'impact de contraintes de mémorisation sur les architectures pour différentes applications en traitement du signal et de l'image.

Abstract *– Systems supporting DSP applications handle more and more data. That involves an intensive use of the memory which becomes the bottleneck of the system; the memory limits the performances and represents a significant part of the overall consumption. Within the framework of RNRT project ALIPTA, we develop a high level synthesis tool called GAUT and integrate the synthesis of the memory part. We will evaluate the impact of memory constraints on architectures for various DSP applications.*

1. Introduction

La spécification d'un IP de niveau comportemental[1] pour des applications en traitement du signal est déclinée en une spécification comportementale de l'algorithme de traitement à implanter à laquelle sont associées les spécifications flexibles des parties mémorisation et communication [1]. Un composant virtuel de niveau comportemental est donc vu comme la combinaison de trois unités fonctionnelles interagissant : une unité de traitement, une unité de mémorisation et une unité de communication. Habituellement, lors de l'écriture de la spécification comportementale, la recherche des parallélismes à grain fin dans les traitements est opérée en premier lieu afin de les exploiter le mieux possible [2]. L'outil de synthèse d'architecture assure l'optimisation des chemins de données tandis que les paramètres des spécifications des autres parties doivent pouvoir être adaptés aux besoins des différentes solutions d'architectures générées et aux contraintes du système. En fonction des paramètres et de la contrainte temporelle imposés par l'utilisateur du composant virtuel, l'outil de synthèse d'architecture aura la tâche d'allouer un nombre adapté de composants de traitement et de les ordonnancer. Néanmoins, si aucune précaution n'est prise à l'égard des communications de l'unité de traitement (E/S et mémorisation), la solution générée par l'outil, bien qu'autorisant un parallélisme des traitements optimal en regard de la contrainte de temps imposée, sera vraisemblablement inexploitable en pratique car la disponibilité au moment voulu des données entrantes et sortantes risque de ne pouvoir être satisfaite. La spécification des unités de traitement, de mémorisation et de communication, doit donc être opérée de manière unifiée en tirant partie des différents schémas de réalisation des traitements, de la répartition des E/S, des accès mémoire, etc.. La structure de la mémoire de travail doit être adaptée au parallélisme de données potentiel requis par les traitements. Sachant que les outils actuels ne synthétisent malheureusement pas encore l'unité de mémorisation, plutôt que de subir des contraintes imposées a posteriori par l'unité de traitement, la méthode consiste à présenter, dès la spécification comportementale, une architecture mémoire et une distribution des données mémoire contraignant partiellement la séquence de traitement. La solution retenue doit favoriser la répartition temporelle des accès afin d'éviter les pics d'accès mémoire.

Dans cet article, nous présentons un état de l'art sur les techniques d'ordonnancement en section 2. Nous introduisons la stratégie globale de conception mémoire en section 3 et détaillons l'ordonnancement sous contrainte de mémorisation mise en œuvre en section 4. Des solutions architecturales pour des applications en TDSI sont présentées et comparées en section 5 avant de conclure.

2. Techniques d'ordonnancement

Un des facteurs importants qui affecte le coût d'une architecture mémoire est l'ordre relatif des accès mémoire contenus dans la spécification. La plupart des méthodes d'ordonnancement prenant en compte la partie mémoire essaient de réduire le coût mémoire en estimant les besoins en terme de nombre de registres pour un ordonnancement donné (seulement pour les scalaires). Les quelques exceptions sont

[1] Ce travail s'inscrit dans le cadre du projet RNRT ALIPTA portant sur la définition et l'application d'une méthodologie de développement pour les (IP) Intellectual Property de niveau comportemental dans les applications de télécommunication

les "stream schedulers"[3], les "rotations schedulers" [4] et les "percolation schedulers"[5]. Ils ordonnancent les accès et les opérations comme un contexte de compilation, en incluant des modèles temporels précis des accès mémoires afin d'améliorer les performances. Ces techniques tentent de réduire le nombre global d'accès mémoire et le nombre d'accès en parallèle. Cependant, elles ne tiennent pas compte des données accédées simultanément afin de faciliter la tâche postérieure d'assignation de registres et mémoires.

Dans le contexte de la synthèse d'architecture, quelques méthodes d'ordonnancement prennent en compte les aspects mémoire. Dans [6], les accès mémoires sont représentés comme étant des opérations multi-cycles dans un CDFG. Les nœuds mémoires sont ordonnancés comme des nœuds opérations en prenant en compte les conflits d'accès aux données. Dans les travaux proposés en [7], un premier ordonnancement de type "force directed scheduling" est réalisé sur un DFG et les accès mémoire sont ensuite ré-ordonnancés en fonction d'une sélection et d'une allocation mémoire visant à réduire le coût global de la mémorisation.

Notre étude se propose de développer une méthodologie visant à intégrer les contraintes liés à la mémorisation dans le flot de synthèse de l'outil GAUT. Nous proposons également d'intégrer les durées de vie des variables pour leur mémorisation ce qui n'est pas réalisé dans [6] et de réduire la complexité de l'ordonnancement par rapport à la technique développée en [7]. L'objectif est de trouver une solution architecturale répondant aux contraintes de l'application et à un compromis entre le temps de conception et la qualité de la solution.

3. Stratégie globale

L'unité de mémorisation, son organisation et son architecture (hiérarchie, cache, nombre de bancs, nombre de ports…) ainsi qu'aux données qui y sont placées et transférées vont fortement conditionner les performances des systèmes. Aussi, nous proposons une stratégie de conception [8], décomposée en trois phases de granularités décroissantes, visant à intégrer la mémoire autour de la synthèse d'architecture et de l'outil GAUT [9].

La première étape se situe en amont de la synthèse d'architecture et de la compilation ; elle comporte des méthodes indépendantes de l'architecture ciblée ; des transformations de code permettent éventuellement d'optimiser le nombre de transferts. On peut ensuite définir la hiérarchie mémoire en nombre de niveaux de hiérarchie, en taille et largeur de chaque niveau et en nombre de ports : les choix s'effectuent par rapport à des fonctions de coût s'appuyant sur des modèles caractérisant les mémoires en fonction de leurs paramètres architecturaux, algorithmiques et technologiques (librairie mémoire). Les structures de données sont ensuite distribuées à travers la hiérarchie et les différents bancs de l'unité de mémorisation. Cette étape de distribution des structures permet de définir un *mapping mémoire*, qui consiste à affecter à chaque structure de données un banc mémoire et une adresse.

Il s'agit dans l'étape suivante d'intégrer les contraintes liées à ce *mapping mémoire* durant les phases classiques de la synthèse architecturale de l'unité de traitement. Cette nouvelle contrainte sera prise en compte lors des différentes étapes de la synthèse et principalement lors de l'ordonnancement des opérations. En effet, l'ordonnancement de l'application doit tenir compte d'éventuels conflits d'accès aux données en fonction de leur distribution dans la hiérarchie et du type de mémoires sélectionnées. L'ordonnancement associé à la prise en compte du mapping mémoire permettra de réaliser un compromis entre les optimisations de l'unité de traitement (en conservant les optimisations possibles avec les nouvelles contraintes) et l'unité de mémorisation (en réduisant le nombre de ports et le nombres de bancs mémoire). Ces optimisations permettent d'obtenir une architecture réaliste et performante.

Enfin, après la synthèse d'architecture, il faut s'intéresser au placement des données scalaires que l'outil a défini comme étant des données à placer en mémoire et définir les générateurs d'adresses associés à chaque mémoire. Les générateurs d'adresses pourront être optimisés en utilisant des techniques d'encodage afin diminuer le nombre de transitions sur les bus d'adresses.

4. Ordonnancement sous contrainte de mémorisation

Avant de contraindre le cœur de synthèse de l'outil GAUT, il faut définir deux choses : une pré-assignation des données en mémoire et une caractérisation des composants mémoire dans lesquels ont été assignées les structures de données (sous la forme de bibliothèque de composants mémoire).

La pré-assignation des données en mémoire sera exprimée par le *mapping mémoire* qui est un fichier de contraintes contenant les structures à placer en mémoire, le numéro de bancs dans lequel elles sont distribuées et éventuellement leurs adresses comme représentées en Fig. 1.

```
X,   numbancs,   adresse, taille;
H,   numbancs,   adresse, taille;
```

Fig. 1 : *Mapping mémoire* pour LMS 4 points.

Les structures de données peuvent être éclatées de manière à contrôler la distribution et le placement des données comme représenté en Fig. 2.

```
x(0),  numbancs,   adresse, ;
x(1),  numbancs,   adresse, ;
x(2),  numbancs,   adresse, ;
x(3),  numbancs,   adresse, ;
h(0),  numbancs,   adresse, ;
h(1),  numbancs,   adresse, ;
h(2),  numbancs,   adresse, ;
h(3),  numbancs,   adresse, ;
```

Fig. 2: *Mapping mémoire* avec structures éclatées.

Pour caractériser les composants mémoire dans les librairies mémoire, il faut d'abord définir les paramètres qui seront utiles lors du cœur de la synthèse d'architecture. Les principaux paramètres sont

-les modes d'accès (burst, aléatoire).

-les temps d'accès mémoire / mode d'accès.

-la taille mémoire.

-le nombre et type de ports.

-le coût en temps et puissance par mode d'accès mémoire.

Dans l'ordonnancement proposé, les données en entrée des opérations peuvent provenir de mémoires internes contenant les données nécessaires à l'unité de traitement. Plusieurs opérations peuvent accéder à la même mémoire à un instant donné. La liste des opérations exécutables est d'abord triée, de façon classique, suivant la mobilité des opérations ; mais le choix de l'opération à ordonnancer s'effectue maintenant suivant un critère d'accessibilité des opérations aux variables. Ce critère est défini de la manière suivante : toutes les opérations exécutables dont les nœuds variables associés sont stockés en registre ou dans une mémoire accessible vérifient le critère d'accessibilité des opérations aux variables. Une mémoire est dite accessible, si à l'instant de l'accès elle possède au moins un port d'accès libre. Donc, pour chaque opération sollicitant des données placées en mémoire, et suivant sa priorité, on vérifie que les données à accéder ne sont pas placées dans des bancs mémoires occupés. Lorsqu'une opération exécutable vérifie le critère d'accessibilité, il faut remettre à jour la liste des mémoires libres. Toutes les opérations ne répondant pas au critère d'accessibilité sont retirées de la liste des opérations exécutables. Par exemple, si une mémoire est occupée, alors une opération exécutable nécessitant un accès à cette mémoire, bien que prioritaire dans la liste, ne pourra être ordonnancée. Les opérations exécutables sont ordonnancées en fonction des opérateurs arithmétiques disponibles. Une fois les opérations ordonnancées, il faut remettre à jour la liste des opérations exécutables, la liste des opérateurs libres et la liste des mémoires accessibles.

5. Applications

5.1 OFDM

Le laboratoire LESTER participe au projet CPER de développement d'une plate-forme de radiocommunication. Le développement de cette plate-forme s'inscrit dans l'optique d'assurer au plus grand nombre d'utilisateurs les débits les plus importants possibles tout en optimisant l'utilisation du spectre électromagnétique. Dans cette optique, les applications mettant en œuvre des systèmes de communication MIMO combinés à des modulations de type OFDM sont privilégiées. Le LESTER, dans le cadre de ce projet devra fournir des solutions architecturales pour les algorithmes à implémenter dans la plate-forme. La multiplicité de la modulation OFDM est basée sur un algorithme de FFT. Aussi, nous montrerons l'influence du placement des données en mémoire sur la solution architecturale issue de la synthèse de haut niveau réalisée par l'outil GAUT pour une spécification de niveau comportemental d'une FFT 1024 points. En fonction des contraintes temporelles (cadence de l'application et temps d'accès des mémoires) et un placement en mémoire prédéfini (nombre de bancs mémoire, placement des structures de données dans les bancs mémoires et adresses de ces structures) nous pouvons évaluer l'impact sur l'architecture de l'unité de traitement réalisant la FFT 1024 points. La Fig.3 présente les résultats de la synthèse d'architecture pour différentes contraintes de *mapping mémoire*. Les synthèses sont réalisées à partir des contraintes suivantes : deux bancs mémoire sont spécifiés avant synthèse, et seule la distribution des échantillons réels et imaginaires diffère. Nous avons effectué 7 distributions des échantillons différente afin de visualiser l'impact au niveau de la complexité en terme de surface (en nombre de CLBs). Les synthèses sont réalisées en utilisant une bibliothèque caractérisée pour les FPGA virtex_E400.

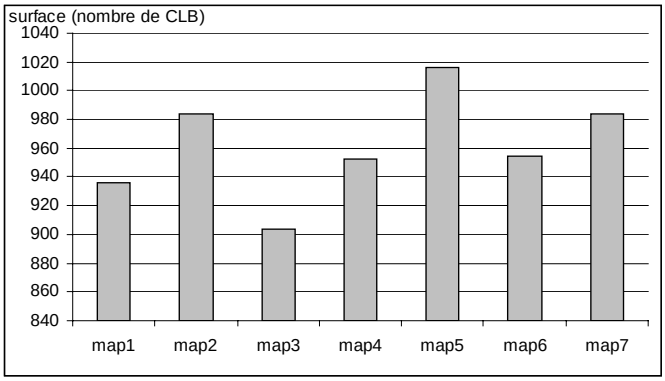

Fig. 3 : surface en fonction de la distribution des données

L'expérience montre que la surface des solutions architecturales varie en fonction de la distribution des données à travers l'architecture mémoire prédéfinie : ici, nous obtenons une variation de 11% du nombre de CLBs. Par ailleurs les résultats fournis dans [10] font ressortir l'influence de la distribution sur la consommation de l'unité de traitement, avec des variations allant jusqu'à 50%.

5.2 Annulation d'écho acoustique

La stratégie d'annulation d'écho acoustique repose sur une estimation du canal acoustique de l'écho en identifiant la réponse impulsionnelle du bouclage acoustique qui est retranché du signal émis. La réponse varie au cours du temps et impose un filtrage adaptatif. Deux stratégies ont été développées pour annuler l'écho acoustique : les algorithmes de moindres carrés récursifs et les algorithmes utilisant le gradient stochastique. Les algorithmes de moindres carrés récursifs sont optimaux au sens de la convergence et ont une complexité très élevée. Les algorithmes de gradient stochastique ont une complexité relativement faible et des vitesses de convergence médiocre. Il existe plusieurs algorithmes de gradient stochastique ; nous allons comparer dans ce paragraphe, les algorithmes de filtrage adaptatif sur

32 points LMS, BLMS (bloc de 4) et GAL (10 cellules). Nous comparerons les ressources utilisées par les unités de traitements et la complexité des unités de mémorisation nécessaires pour garantir les contraintes temporelles.

Nous avons réalisé des synthèses pour différentes contraintes de temps et déterminer les architectures mémoire (en nombre de bancs mémoire) et la distribution des données dans ces bancs mémoire qui garantissent les fréquences de fonctionnement pour chaque solution.

Dans la Fig. 4,et la Tab. 1, nous comparons, pour différentes contraintes de cadence (1 Mhz et 2 Mhz), la complexité des unités de traitement, en nombre de cellules logiques (synthèses d'architecture réalisées pour une cible FPGA virtexE_400) et celles des unités de mémoires en nombre d'accès mémoire, en nombre et taille de bloc de ram, et en nombre de FSM permettant de piloter les accès mémoire en unité de traitement et unité mémoire. Cette expérience permet de comparer la complexité des solutions architecturales pour les trois algorithmes d'annulation d'écho acoustique.

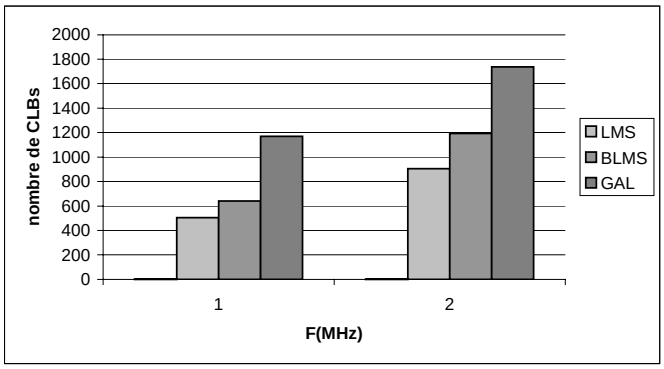

Figure 4 : complexité des unités de traitement synthétisées

TAB. 1 : complexité des unités de mémorisation

		nombre de points mémoire	nombre d'accès mémoire	mapping mémoire (nb bancs)
1 MHz	LMS	65	160	3
	BLMS	135	842	4
	GAL	217	575	8
2 MHz	LMS	90	210	4
	BLMS	132	694	6
	GAL	154	422	10

On retrouve la complexité des algorithmes en comparant les surfaces des unités de traitement de la Fig. 4 ; du LMS, le plus simple au GAL le plus complexe. La complexité des unités de mémorisation peut s'évaluer par le nombre de bancs mémoire définis par le *mapping mémoire* permettant de respecter la contrainte temporelle. Pour les deux contraintes temporelles définies dans la Tab. 1, le filtre LMS est celui qui nécessite le moins de ressources. Le Filtre BLMS nécessite moins de nombre de points de mémorisation mais plus d'accès mémoire que le Filtre GAL. Le filtre BLMS reste cependant moins complexe que le filtre GAL car le nombre de mémoire à mettre en œuvre pour tenir la contrainte de temps est toujours plus faible.

6. Conclusion

La stratégie développée alliée à la technique d'ordonnancement sous contrainte de *mapping mémoire* mise en œuvre lors de la synthèse d'architecture permettent de déterminer l'architecture mémoire la plus adaptée à l'application. Les résultats fournis montrent l'impact de l'architecture mémoire sur la complexité de l'unité de traitement. Par ailleurs, cette stratégie permet d'évaluer la qualité de différents algorithmes en termes de complexité de leurs unités de traitement et de mémorisation. Nous apporterons des améliorations à l'ordonnancement en permettant l'anticipation des lectures de données afin de réduire les pénalités temporelles au niveau de l'unité de mémorisation. De plus, nous mettrons en œuvre une technique permettant de réduire la complexité algorithmique de notre ordonnancement afin d'accélérer le processus de synthèse.

Références

[1] R. Airiau, A. Carer, E. Casseau, E. Martin, O. Sentieys,"*Méthodologie de conception de composants virtuels pour les applications de TDSI*, " Workshop sur l'adéquation algorithme architecture, pp. 65-69, jan 2000,

[2] G Savaton, E. Casseau, E. Martin, C. Lambert-Nebout, "*Composants virtuels comportementaux pour applications de compression d'images*," *in proceeding of GRETSI'01*, septembre 2001.

[3] W. Verhaegh, P. Lippens, E. Aarts, J. Korst, J. van Meerbergen, A. van der Werf,"Improved Force-Directed Scheduling in High-Throughput Digital Signal Processing", *IEEE Trans. on Computer-aided design*, Vol.14, No.8, p 945-960, Aug. 1995.

[4] N. Passos, E. Sha, L-F. Chao, "Multi-dimensional interleaving for time-and-memory design optimization", In Proc. *IEEE Int. Conf. On Computer Design*, pp.440-445, Oct. 1995.

[5] A. Nicolau and S. Novack.,"Trailblazing A hierarchical approach to percolation scheduling," In *Proc of ICPP*, St. Charles, pp 120-124, 1993.

[6] H. Ly, D. Knapp, R. Miller, D. McMillen,"Scheduling using Behavioral Templates," In *Proceeding of Design Automation Conference*, pp 101-106, June 1995.

[7] J. Seo, T. Kim, P. Panda,"An Integrated Algorithm for Memory Allocation and Assignment in High-Level Synthesis," In *Proceeding of Design Automation Conference*, pp 608-611, June 2001.

[8] G. Corre, N. Julien, E. Senn, E. Martin,"Intégration de la synthèse mémoire dans l'outil de synthèse d'architecture GAUT low power," *In Proceedings of JFAAA*, Dec 2002.

[9] http://lester.univ-ubs.fr:8080/

[10] G. Corre, N. Julien, E. Senn, E. Martin,"Ordonnancement sous contrainte de mémorisation : une optimisation efficace des ressources lors de la synthèse d'architecture," *In Proceedings of FTFC*, pp 174-152, May 2003.